ThingVisor Factory: Thing Virtualization Platform for Things as a Service

Kenji Kanai
Waseda Research Institute for Science and Engineering
Waseda University
Tokyo, Japan
k.kanai@aoni.waseda.jp

Hidenori Nakazato
Dept. of Computer Science and Comm. Engineering
Waseda University
Tokyo, Japan
nakazato@waseda.jp

Hidehiro Kanemitsu
Dept. of Computer Science
Tokyo University of Technology
Tokyo, Japan
kanemitsuh@stf.teu.ac.jp

Andrea Detti
Electronic Engineering Dept.
University of Rome Tor Vergata, CNIT
Rome, Italy
andrea.detti@uniroma2.it

ABSTRACT

In order to provide interoperability of cross-domain IoT applications involving different IoT platforms, the authors previously proposed a virtual IoT system called VirIoT. The proposed system is composed of two functionalities: ThingVisor and vSilo, and it aims at decoupling IoT device providers and IoT application developers. ThingVisor enables to produce virtual IoT devices, or Virtual Things, from physical IoT devices for sharing the physical devices among cross-domain IoT applications. In addition, vSilo enables to bridge between such Virtual Things and IoT applications for the interoperability of cross-domain IoT devices. In this paper, in order to enhance the VirIoT system, we propose ThingVisor Factory that helps to design ThingVisors in a user-friendly way and deploy them on demand autonomously by following container orchestration methodologies, such as Kubernetes. ThingVisor Factory is based on two concepts: dataflow programming-based Graphical User Interface (GUI) and service function chaining-based ThingVisor development.

CCS CONCEPTS

Networks → Network services → In-network processing

KEYWORDS

IoT virtualization, Thing Hypervisor, Service function chaining, IoT platform

ACM Reference format:

Kenji Kanai, Hidenori Nakazato, Hidehiro Kanemitsu, Andrea Detti. 2020. ThingVisor Factory: Thing Virtualization Platform for Things as a Service. In *Cloud Continuum Services for Smart IoTSystems (CCIoT '20), November 16–19, 2020, Virtual Event, Japan. ACM, New York, NY, USA, 6 pages.* https://doi.org/10.1145/3417310.3431399

CCIoT'20, November 16-19, 2020, Virtual Event, Japan

ACM ISBN 978-1-4503-8131-4/20/11...$15.00
https://doi.org/10.1145/3417310.3431399

1 Introduction

Thanks to evolution of wireless sensor networking and cloud computing, Internet of Things (IoT) has been not only collecting interest in academia but also applied in business areas [1, 2]. IoT technologies are expected to contribute to accelerating development of smart cities, while smart city application developers and providers require to simplify IoT application development and support large scale IoT deployment for reduction of expenditure and longer sustainability.

To address this demand, currently, IoT cloud services and IoT platform pay much attention to smart cities application developers. The IoT cloud services, such as Google Cloud IoT, AWS IoT and Microsoft Azure IoT, provide cloud computing platform and network accessibility between IoT devices and cloud servers to the developers. In addition, the IoT platforms, such as oneM2M [3] and FIWARE [4], arrange IoT data sharing platforms. Such IoT platforms can exchange IoT data between cross-domain smart cities applications via specific IoT brokers, such as Mobius Broker [5] and Orion Context Broker [6], by encapsulating semantic IoT data model, such as oneM2M and Next Generation Service Interface v2 (NGSIv2). For the developers, preparing such environments from scratch is a heavy burden and consumes large expenditure. Therefore, Infrastructure as a Service (IaaS) solution is current main streams for the developers.

However, such IoT platformers construct the IoT platforms as isolated "silos" containing both the infrastructures and the IoT software services. In addition, in the isolated silo environments, the IoT platforms do not provide the interoperability of cross-domain IoT applications among different IoT platforms. This situation

prevents entering brand new application developers to deploy their smart IoT applications and includes a risk of opposing growth of IoT business, such as development of smart cities.

To address this fact, the authors of this paper tackle the research project named "Fed4IoT [7]" which is a Research and Innovation Action jointly funded by the European Commission and Japan's Ministry of Internal Affairs and Communications (MIC). Fed4IoT aims at realizing the federation of IoT and cloud infrastructures to provide scalable and interoperable smart city applications. Fed4IoT project will not standardize a brand new IoT platform itself but re-use the concepts of IoT platforms and virtually integrate the cross-domain IoT platforms by adapting the concepts of visualization technologies, such as computer virtualization and network virtualization.

In our previous work [8], we have proposed the key concepts of Fed4IoT, such as IoT Virtualization (*VirIoT*), Thing Hypervisor (*ThingVisor*) and Virtual Silo (*vSilo*). As presented in the same paper, VirIoT enables to decouple the IoT platformers (e.g., IoT infrastructure providers) from both IoT application developers and providers. Thus, the IoT application developers can develop their own IoT services by sharing the IoT data produced from the cross-domain IoT platforms. In addition, the IoT application providers (or tenants) can quickly deploy their IoT applications in cross-domain sites (e.g., smart cities) without any adaptation of domain-specific IoT infrastructures because VirIoT absorbs such differences.

To enable such decoupling, Fed4IoT introduces two unique concepts: ThingVisor and vSilo. ThingVisor stands for the IoT application developers and enables to produce virtual IoT devices, or *Virtual Things*, from physical IoT devices in order to share the IoT devices among cross-domain IoT services, and vSilo stands for the IoT application providers and enables to bridge between such Virtual Things and IoT applications in order to achieve the interoperability of cross-domain IoT devices.

In this paper, in order to enhance the concepts of VirIoT, in particular ThingVisor, we propose *ThingVisor Factory* that helps to design on-demand ThingVisors in a user-friendly way and deploy them autonomously by considering their networking and computing demands in a cloud/edge/fog computing manner. To address this issue, ThingVisor Factory is based on two concepts: dataflow programming-based Graphical User Interface (GUI) and service function chaining-based ThingVisor development. It should be noted that ThingVisor Factory does not require any modifications to VirIoT (and other IoT platforms) and can be used as middleware to develop and deploy ThingVisors. ThingVisor Factory is closely related to one of the deliverables of our Fed4IoT project, and the paper summarizes the content of Deliverable 3.2 [7].

2. Related Work

2.1 Virtualization Technologies

As reported in the survey [9], many researches have been conducted on object (or "Thing") virtualization which represents the methodology of how to map physical devices into virtual spaces, and this concept is quite similar to cyber-physical system. According to the same survey, authors of [9] introduced four types association between real and virtual objects: one-to-one, one-to-many, many-to-one, and many-to-many association. The one-to-one association indicates that a single physical object (or IoT device) produces only one virtual object, and oneM2M and FIWARE handle the IoT data in such association manner. The one-to-many association indicates that a single physical object produces multiples virtual objects, and, the IoT-A project [10] and the COMPOSE project [11] attempted to develop such functionalities in order to facilitate the service orchestration. Alternatively, the many-to-one association denotes that multiple physical objects are aggregated to a single virtual object, and the SENSEI project [12] studied on such association where heterogenous sensors and actuators are integrated into a homogenous (virtual) device for efficient management and operation. Furthermore, the many-to-many association represents a hybrid case between many-to-one and one-to-many associations, and the iCore project [13] provided such functionality of interoperable physical objects and/or virtual objects in order to satisfy diverse application requirements. The iCore architecture can reuse (or share) the physical objects among virtual objects, and vice versa.

Inspired by the research projects and efforts, the Fed4IoT project proposes virtual IoT platform (VirIoT) [8] enables to interoperable among cross-domain IoT devices, IoT platforms and IoT applications by adapting novel visualization technologies, such as container-based virtualization and networking softwarization. Although the concept of Thing virtualization is based on the related research projects [10-13], VirIoT introduces the concept of Thing virtualization, named "*Thing Hypervisor* (*ThingVisor*)", by referring a more cloud-native virtualization scheme in order to interoperate other cloud-native platforms, including FIWARE-native FogFlow [15], and other IoT platforms.

2.2 Service Function Chaining

As presented in [8], service function chaining is one of candidate technologies to develop ThingVisors. Service function chaining is one of network virtualization (or softwarization) technologies regarding Software Defined Networking (SDN) and Network Function Virtualization (NFV). Because the service function chaining can be realized in-network processing, it is possible for network operators to manage networking and computing resources efficiently and flexibly. The service function chaining is also one of promising solutions in order to realize Thing virtualization in the Fed4IoT project, while operation and management of service function chaining heavily depends on communication protocols. In order to realize the service function chaining, there are three candidates of communication protocols, such as P2P over IP, Pub/Sub over IP and Information Centric Networking (ICN). One of challenging issues for service function chaining is routing resolution.

In P2P over IP model, the service function chaining is operated by the SDN manner, such as OpenFlow. The service functions (or virtualized network functions) are identified by using specific tags

called Network Service Header (NSH) tags. The NSH tag is embed in IP packet header, and IP routers forward the IP packets according to the NSH tag. The architecture of this service function chaining is standardized as IETF RFC7665 [16] and RFC 8300 [17].

In Pub/Sub model which is proposed in the previous work [14], unlike the P2P over IP model, the routing for the service function chaining is handled on the application layer. The service functions are identified by topic names for Pub/Sub communications, such as MQTT and Apache Kafka. The service function subscribes a specific topic name to receive a required data and publishes an alter topic name to produce a processed data. In this model, because the routing can be managed on the application layer, it is easy to implement, but, load balancing of Pub/Sub brokers is essential.

In ICN [18] model, this case, the communication model is completely different from the previous two models. Because ICN routing is ideally resolved by not IP address but content name, the communication model indicates request-response-type communication. ICN service function chaining [19, 20] is required to the management of routing table called forwarding information base (FIB) on ICN router level like P2P base and the management of interest name on the application layer like a Pub/Sub base. Although, ICN service function chaining is a challenging topic, potential ICN capabilities, such as in-network caching and in-network processing, provide more efficient and flexible operation to the service function chaining.

3. Thing Virtualization

As presented in Introduction, Thing Virtualization is a unique and important concept of VirIoT. For IoT application developers, processed data (or knowledge) is important rather than the raw IoT data produced by physical IoT devices (e.g., Real Things). In VirIoT, such knowledge is defined as "Virtual Thing" and "Thing Hypervisor (ThingVisor)" produces and manages diverse such Virtual Things instead of IoT application developers. As Similar to Hypervisor of computing virtualization, ThingVisor can conceal different hardware configurations of Real Things, including network configurations and provide appropriate IoT data processing, like data copying, fundamental statistical analysis and complex image processing, instead of IoT application (i.e., ThingVisor create Virtual Things from Real Things). Although FogFlow is an attractive solution to realize ThingVisors, we adopt service function chaining technology to realize ThingVisors. By using ThingVisors, IoT application developers need not to pay attention of conditions of Real Things and data processing of Real Things, and just access to Virtual Things in order to retrieve require IoT data or knowledge. Thus, ThingVisor the key concept in order to decouple Thing providers and IoT application developers, and "ThingVisor developers" may do their businesses such as "Things as a Service".

4. VirIoT ThingVisor Factory

4.1 Concept of VirIoT ThingVisor Factory

VirIoT ThingVisor Factory is a platform that can provide functionalities for designing, developing, and deploying ThingVisors on-demand for IoT developers, or tenant. Through VirIoT ThingVisor Factory, developers can interactively design and develop their own ThingVisors, including "private" ThingVisors that produce tenant-specific Virtual Things, such as a face detection ThingVisor for a specific person. In addition, developers need not pay attention to installation of IoT devices as long as they exist in the VirIoT environment, and to deployment of required data processing functionalities over the Internet.

One of the challenging issues in VirIoT ThingVisor Factory is how to deal with diverse requirements from developers, by satisfying their networking and computing demand in a user-friendly way, (e.g., autonomous, instinctive, and interactive way.) To address this issue, VirIoT ThingVisor Factory is equipped with two functionalities: dataflow-programming-based graphical user interface (GUI) and service function chaining-based ThingVisor development.

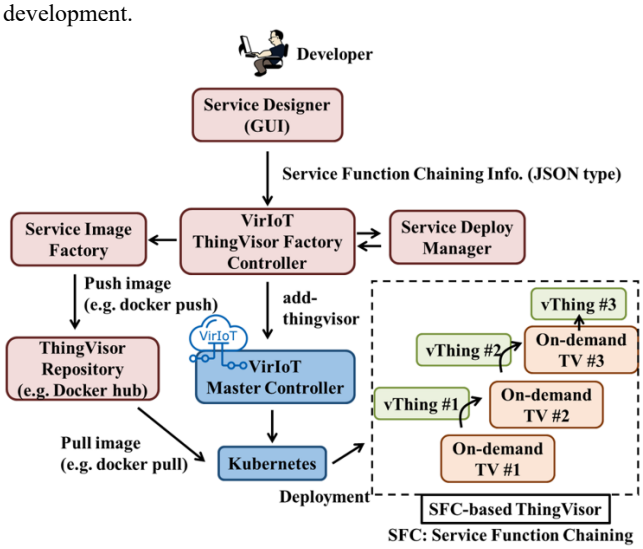

Figure 1: Architecture of VirIoT ThingVisor Factory

4.2 Architecture of VirIoT ThingVisor Factory

The architecture of VirIoT ThingVisor Factory is shown in Figure 1, where red function blocks represent the components of VirIoT ThingVisor Factory and blue function blocks represent components of VirIoT system. As shown this figure, VirIoT ThingVisor Factory provides modules that complement VirIoT functionality, and it supports the implementation and deployment of a chain of ThingVisors on demand. In order to provide a user-friendly platform and autonomous management, VirIoT ThingVisor Factory mainly composes three key functionalities: Service Designer, Service Image Factory and Service Deploy Manager.

Through the VirIoT ThingVisor Factory, developers, or tenants, can develop their own ThingVisors. This operation is done from the GUI provided by Service Designer. In VirIoT ThingVisor Factory,

Virtual Things are defined as outputs from a chain of service functions created by dataflow programming. After the developer completes designing ThingVisors and their chains, Service Designer outputs JSON serialized service function chaining information corresponding to the designed chains of ThingVisors. The service function chaining information indicates a specification of a ThingVisor chain and is composed of the information of required service functions, or ThingVisors, and connectivity among the ThingVisors. Based on the service function chaining information, Service Image Factory creates Docker Images for required ThingVisors and pushes those images to a ThingVisor repository, such as Docker Hub. Meanwhile, Service Deploy Manager determines the deployment plan of dockerized service functions, by considering networking and computing conditions of network nodes (e.g., Kubernetes nodes).
These functionalities are operated with APIs provided by VirIoT ThingVisor Factory Controller. After all preparation is done (i.e, service functions are stored on Docker Hub, and deployment plan of them are defined), VirIoT ThingVisor Factory Controller invokes an "add ThingVisor" command on the VirIoT Master-Controller in order to deploy the ThingVisor on the platform.

In the next three Sections, we present here the detailed description of three key components of VirIoT ThingVisor Factory.

4.3 Service Designer

The first key component of our VirIoT ThingVisor Factory is the Service Designer. Because VirIoT ThingVisor Factory is required to develop on-demand ThingVisor instinctively and interactively, Service Designer provides dataflow programming-based GUI as shown in Figure 2. As shown in the figure, Service Designer abstracts the common functionalities as "service function blocks" and visualizes as colorful "blocks." Service Designer is similar to (and based on) Node-RED, but, unlike Node-RED, Service Designer can specify the network connectivity between service function blocks. More specifically, the developer can specify the communication protocols for service function chaining, such as P2P, Pub/Sub and ICN as mentioned in the related work. This is the biggest different aspect against Node-RED, and VirIoT ThingVisor Factory aims at network-wide deployment (and operation) of service functions while Node-RED can only deploy (and operate) on the local Node-RED environment.

In order to design ThingVisor instinctively and interactively, Service Designer abstracts the typical service functions as blocks. As shown in Figure 2, Service function blocks compose "sensor," "service function," "connector," and "program". First, a sensor block provides a functionality of retrieving Real Things and Virtual Things such as temperature values, surveillance camera images or video. The sensor block contains the information of how to access Thing (e.g., API, broker information, and protocol), Thing ID and meta data of Thing (e.g., geolocation). Second, a service function block provides a functionality of IoT data processing, such as statistical analysis and image processing. This block contains the information regarding data processing engine, such as input and output data formats (or types) and docker image name. It also includes meta data of service function (e.g., description of service function and author's name). Next, a connector block, which is one of important blocks, provides a definition of network connectivity among blocks. This block contains the information of communication protocols, such as Pub/Sub or ICN, and it also contains the information of input and output functions in order to describe the chaining operation of service functions. At this moment, the developer can specify topic name or interest name to publish Virtual Thing when the developer selects Pub/Sub or ICN as a communication protocol. Finally, a program block provides a functionality where the developer can write his/her own functionality in specific programming languages, such as Python, similar to Node-RED (e.g., Node-RED allows programmers to write programs in JavaScript).

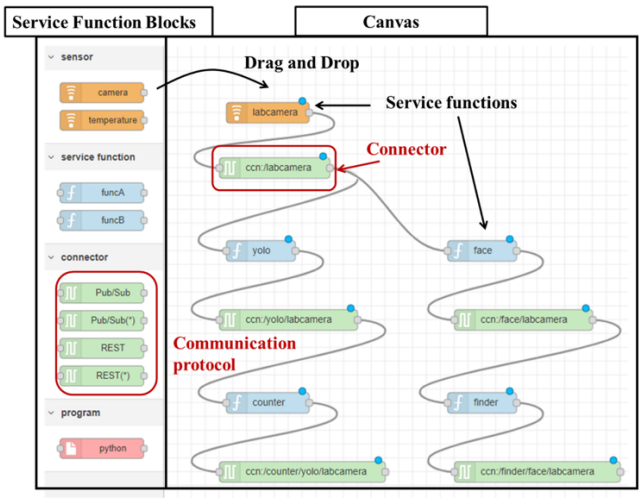

Figure 2: GUI of Service Designer

By drag and drop operations, the developers can select the service functions, and by connecting the blocks with lines, the developers can define the service functions as service function chainings. Thus, Service Designer can provide simple, instinctive and interactive platform to design on-demand ThingVisors. After the developer designs ThingVisor, Service Designer generates and outputs JSON serialized service function chaining information based on the required blocks. By using the service function chaining information, VirIoT ThingVisor Factory prepares docker images regarding the required service functions and determines the deployment plan, including routing resolution, in case of ICN protocol.

4.4 Service Image Factory

The second key component of the VirIoT ThingVisor Factory is the Service Image Factory. Service Image Factory mainly provides a functionality of automatic dockerization of service functions if necessary. First, Service Image Factory parses the service function chaining information produced by Service Designer and figures out whether the service functions need to be dockertized or not. The simplest case is that the developer only selects pre-defined (or pre-developed) service functions. In such case, service functions, including communication protocol, have already dockerized, and

Service Image Factory just modifies operations of service functions by only changing a configuration level, such as changing the names of topics, transmission interval or other parameters regarding data processing. When developers request to compose their own service functions which specified by "program block" in Service Designer, Service Image Factory parses their requests and generates docker images, including network functionality (i.e., communication protocol), according to their source codes. In order to simplify docker image creation, Service Image Factory provides a template of service function. The template guides the developers to program service functions which are certainly dockerized and executed. The template also contains the functions of communication protocols.

After Service Image Factory prepares docker images regarding service functions, Service Image Factory pushes the docker images to ThingVisor repository (e.g., Docker Hub) in order to be able to be pulled from VirIoT master controller.

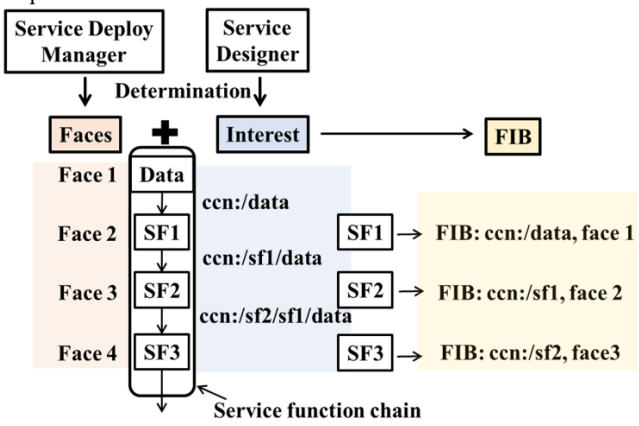

Figure 3: FIB construction in VirIoT ThingVisor Factory

4.5 Service Deploy Manager

The third key component of the VirIoT ThingVisor Factory is the Service Deploy Manager. In order to deploy dockerized ThingVisor autonomously and construct service function chaining, Service Deploy Manager mainly provides two functionalities: determination of optimal deployment plan and route resolution for ICN. In the determination of optimal deployment plan, Service Deploy Manager derives optimized network nodes (e.g., Kubernetes nodes) where the dockerized ThingVisors are deployed. In the operation, Service Deploy Manager considers not only networking and computing resource usages of the network nodes, but also physical locations of the network nodes (i.e., Japan or European and edge or cloud). This optimization is modeled as a workflow scheduling problem and implemented as a workflow engine proposed in the previous work [21].

In the routing resolution for ICN, this functionality is required only for ICN usage case. As described in the related work, ICN is one of good candidates of communication protocols for realizing service function chaining. Unlike TCP/IP, including Pub/Sub model over IP protocol, ICN requires to solve networking routes by using not IP table but forwarding information base (FIB). in ICN-based service function chaining, (autonomous) FIB management is one of challenging issue. To simplify the FIB management, VirIoT ThingVisor Factory adopts a centralized management approach by referring to OpenFlow controller.

Before introducing FIB management, a determination method of FIB is presented. FIB mainly composes (content) name prefix and upstream faces. To determine (or build) FIB in advance, (ideally) ICN routers need to know correct upstream faces corresponding to name prefixes. In general, it is quite difficult to know such information in advance, however, VirIoT ThingVisor Factory can determine the correct upstream faces and name prefixes regarding service function chaining as shown in Figure 3. As shown in the figure, this is because, at first, through Service Designer, the developer designs the service function chaining and specifies the content names corresponding to the service functions. In other words, VirIoT ThingVisor Factory can know the exact interest names transmitted by the service functions in order to retrieve contents. In addition, after the service function chaining is designed, Service Deploy Manager determines the deployment nodes, and this means that VirIoT ThingVisor Factory can know the exact faces corresponding to the content names produced by the service functions. Thus, VirIoT ThingVisor Factory can determine FIB from the specification of service function chaining in advance.

After VirIoT ThingVisor Factory figures out the determination of FIB, it is necessary to populate such FIB information to ICN routers. As mentioned before, by referring architecture of OpenFlow, VirIoT ThingVisor Factory adopts centralized management approach, and its controller distributes FIB information to ICN routers. A preliminary architecture of FIB distribution is shown in Figure 4. As shown in the figure, there are two strategies: ICN/IP Hybrid case and pure ICN case. In the ICN/Hybrid case, data plane is operated over ICN protocol, and control plane is operated over IP protocol. More specifically, FIB distribution can be handled on the control plane, and FIB information multicasts with a topic name specified by Pub/Sub communication. This approach is easy to implement, but control traffic may become large. On the other hand, in the pure ICN case, both data and control planes are operated over ICN protocol, and FIB information is distributed by the ICN manner (e.g., exchange Interest and Data packets). This approach requires more complex implementation such as FIB management for FIB distribution. In addition, realization of push delivery over ICN may be required. After FIB information are successfully distributed, ICN routing for service function chaining is resolved.

Finally, once all preparations are done, VirIoT ThingVisor Factory Controller invokes an "add ThingVisor" command on the VirIoT Master-Controller in order to deploy the ThingVisor on the platform, and ThingVisor processes the IoT data by employing the service function chaining and publishes the proceeded data as Virtual Things to IoT applications.

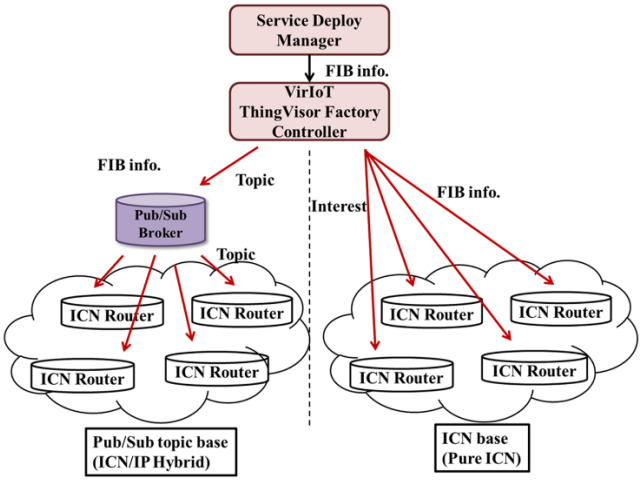

Figure 4: Architectures of FIB information distribution

5. Conclusion and Future Work

In order to provide interoperability of cross-domain IoT applications involving different IoT platforms, we proposed a virtual IoT system (*VirIoT*) in the previous work [8]. VirIoT introduced two unique concepts: *ThingVisor* and *vSilo*. ThingVisor enables to produce virtual IoT devices, or *Virtual Things*, from physical IoT devices in order to share the physical IoT devices among cross-domain IoT applications. In addition, vSilo enables to bridge between such Virtual Things and IoT applications in order to achieve the interoperability of cross-domain IoT devices. In this paper, in order to enhance the VirIoT systems, we proposed *VirIoT ThingVisor Factory* that helps to design on-demand ThingVisors in a user-friendly way and deploy them autonomously by following the container orchestration methodology, such as Kubernetes. To address this issue, VirIoT ThingVisor Factory is composed three key functionalities: *Service Designer*, *Service Image Factory*, and *Service Deploy Manager*.

In future, we will implement VirIoT ThingVisor Factory, prove the concept by deploying actual ThingVisors created by ThingVisor Factory at EU and JP's smart cities and demonstrate the interoperability of our VirIoT systems.

ACKNOWLEDGMENTS

The research leading to these results has been supported by the EU-JAPAN initiative by Horizon the EC Horizon 2020 Work Programme (2018-2020) Grant Agreement Horizon814918 and MIC Ministry of Internal Affairs and Communications "Strategic Information and Communications R&D Promotion Programme (SCOPE)" Grant no. MICJPJ000595 "Federating IoT and cloud infrastructures to provide scalable and interoperable Smart Cities applications, by introducing novel IoT virtualization technologies (Fed4IoT)."

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29(7), pp. 1645–1660, Sept. 2013.

[2] L. Shancang, L. D. Xu, and S. Zhao. "The internet of things: a survey," Information Systems Frontiers, vol. 17(2), pp. 243-259, Apr. 2015.

[3] oneM2M [online]. Available: http://www.onem2m.org/

[4] FIWARE [online]. Available: https://www.fiware.org/

[5] Mobius, oneM2M IoT Server Platform [online]. Available: https://github.com/IoTKETI/Mobius

[6] FIWARE-Orion, Orion Context Broker [online]. Available: https://github.com/telefonicaid/fiware-orion

[7] Fed4IoT project [online]. Available: https://fed4iot.org/

[8] A. Detti, G. Tropea, G. Ro, J. A. Martinez, A. F. Skarmeta, and H. Nakazato, "Virtual IoT Systems: Boosting IoT Innovation by Decoupling Things Providers and Applications Developers", Proc IEEE Global IoT Summit 2019, Jun. 2020.

[9] M. Nitti, V. Pilloni, G. Colistra, L. Atzori, "The Virtual Object as a Major Element of the Internet of Things: A Survey," IEEE Communications Surveys & Tutorials, vol.18, issue: 2, Nov.2015.

[10] Internet of Things Architecture (IoT-A) project (2010 - 2013) [online]. Available: https://cordis.europa.eu/project/id/257521

[11] Collaborative Open Market to Place Objects at your SErvice (COMPOSE) project (2012 - 2015) [online]. Available: https://cordis.europa.eu/project/id/317862

[12] M. Presser, P. M. Barnaghi, M. Eurich, C. Villalonga, "The SENSEI project: integrating the physical world with the digital world of the network of the future," IEEE Communication Magazine, vol.47, Issue: 4, Apr.2009.

[13] iCore, empowering IoT through cognitive technologies (2011 - 2014), [online]. Available: https://ics-iot.weebly.com/icore.html

[14] K. Ogawa, K. Kanai, K. Nakamura, H. Kanemitsu, J. Katto and H. Nakazato, "IoT Device Virtualization for Efficient Resource Utilization in Smart City IoT Platform," IEEE PerCom 2019, Mar.2019.

[15] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities," IEEE Internet of Things Jounal, vol. 5, Issue 2, pp. 696–707, Apr. 2018.

[16] IETF RFC 7665 "Service Function Chaining (SFC) architecture," Oct. 2015

[17] IETF RFC 8300 "Network Service Header (NSH)," Jan. 2018.

[18] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content", Commun. ACM, 55, 1, pp. 117‑124, 2012.

[19] L. Liu, Y. Peng, M. Bahrami, S. Mnatsakanyan, G. Qu, Z. Ye, H. Guo, "ICN-FC: An Information-Centric Networking Based Framework for Efficient Functional Chaining," IEEE ICC 2017, May 2017.

[20] Y. Kumamoto, H, Yoshii, and H. Nakazato, "Real-world implementation of function chaining in Named Data Networking for IoT environment," IEEE CQR, May 2020.

[21] H. Kanemitsu, K. Kanai, J. Katto and H. Nakazato, "A Function Clustering Algorithm for Resource Utilization in Service Function Chaining," IEEE CLOUD 2019, July 2019.